QP: A Heterogeneous Multi-Accelerator Cluster

Michael Showerman[1], Jeremy Enos, Avneesh Pant,
Volodymyr Kindratenko[2], Craig Steffen, Robert Pennington, Wen-mei Hwu
University of Illinois at Urbana-Champaign, Urbana, 61801 IL

Abstract

We present a heterogeneous multi-accelerator cluster developed and deployed at NCSA. The cluster consists of 16 AMD dual-core CPU compute nodes each with four NVIDIA GPUs and one Xilinx FPGA. Cluster nodes are interconnected with both InfiniBand and Ethernet networks. The software stack consists of standard cluster tools with the addition of accelerator-specific software packages and enhancements to the resource allocation and batch sub-systems. We highlight several HPC applications that have been developed and deployed on the cluster. We also present our Phoenix application development framework that is meant to help with developing new applications and migrating existing legacy codes to heterogeneous systems.

Keywords: heterogeneous system, acceleration co-processor, GPGPU, FPGA

1 Introduction

In the past, high-performance computing (HPC) community counted on perpetual CPU performance improvements due to the increase in the clock frequency. However, increasing clock frequency has become very difficult due to leakage current and thermal density. In order to take advantage of shrinking transistor sizes, semiconductor manufacturers have resorted to placing multiple copies of the same processor design on a single die. Four- and eight-core processors are available now, and the trend is to integrate even more cores and auxiliary hardware in a single chip. Consequently, many of the recently deployed HPC systems consist of symmetric multiprocessor (SMP) compute nodes with several multi-core chips in each node.

Application-specific architectures, such as those used to accelerate computer graphics and digital signal processing applications, have evolved to the point when their functionality, while still limited, can be used in many applications to achieve performance levels not attainable even on today's most advanced multi-core chips. As the result, a new trend in HPC is emerging: the use of accelerators to speed up computationally intensive parts of applications. We envision that many HPC system of the near future will consist of heterogeneous nodes built from multi-core processors and accelerators.

The Innovative Systems Laboratory (ISL) at the National Center for Supercomputing Applications (NCSA) developed and deployed a prototype 16-node cluster called QP (for "Quadro Plex") that includes multi-core CPUs and two types of accelerators: graphical processing units (GPUs) and field-programmable gate arrays (FPGAs) [1]. The cluster has been extensively used since 2007 for teaching courses in programming massively parallel systems [2] and for developing and running scientific applications by research groups across the country [3]. In this paper we describe the design of the system, outline our vision of a run-time environment for supporting heterogeneous cluster architectures, and provide examples of a few applications designed to run on the cluster.

[1] Presenting author: mshow@ncsa.uiuc.edu
[2] Corresponding author: kindr@ncsa.uiuc.edu

2 System Architecture

The first step in planning the deployment of an accelerator-enhanced cluster is to determine the design goals regarding the required scale, CPU core to accelerator ratio, FLOP (floating-point operation) to network I/O ratio, and budget. We have chosen a CPU core to GPU ratio of 1:1. This greatly simplifies application development since every application thread that utilizes one CPU core can also have exclusive access to one GPU. We have chosen to have a single FPGA per cluster node due to the hardware availability and cost. As accelerators, such as GPUs, enter the HPC cluster paradigm, it is easy to have enough FLOPs within one host to drive up the penalty of a network bottleneck. Therefore, we have chosen to add InfiniBand network in spite of its relatively high cost as compared to Ethernet.

Figure 1 shows the overall cluster architecture. Each of the 16 compute nodes has two dual-core 2.4 GHz AMD Opteron CPUs, 8 GB of RAM, four NVIDIA Quadro FX 5600 GPU cards (each has 1.5 GB of on-board memory), one Nallatech H101-PCIX FPGA accelerator card (with 16 MB of SRAM and 512 MB of SDRAM), and InfiniBand and Ethernet PCIe network cards (Figure 2). The system occupies four 42U racks, draws under 18 kW of power, and has the theoretical peak performance of 23 TFLOPS in single-precision floating-point arithmetic.

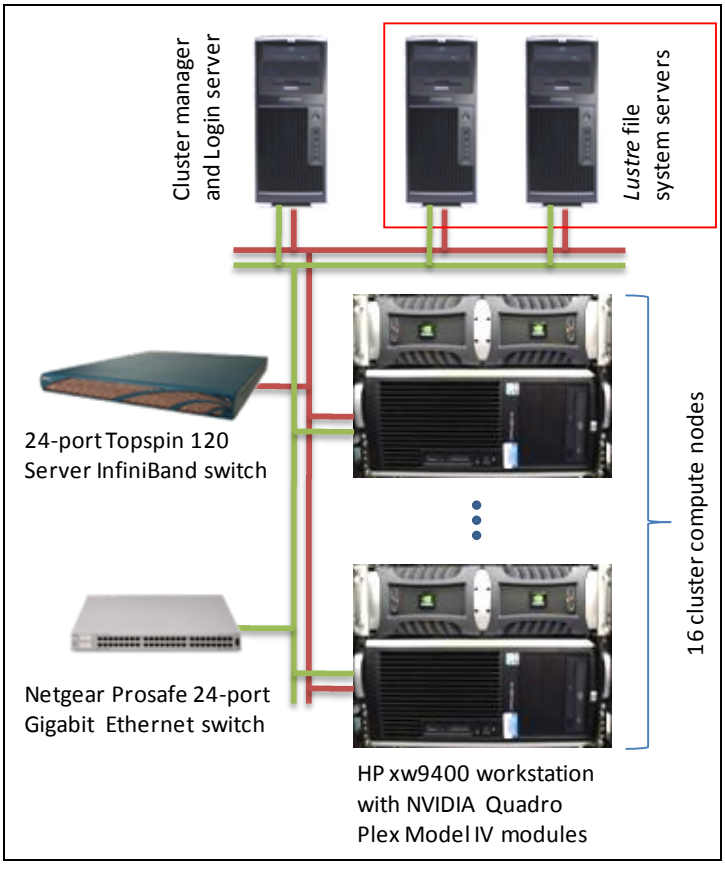

Figure 1. Multi-core/GPU/FPGA cluster architecture.

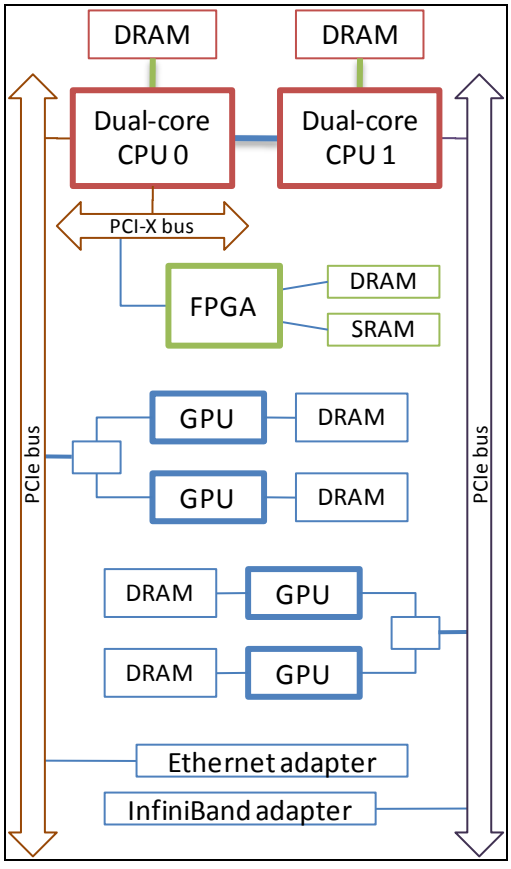

Figure 2. Compute node architecture.

In addition to the 16 compute nodes, additional workstations are used for serving a high-performance file system and as the cluster's "head node." The head node, shown as cluster manager and login server in Figure 1, is an access point or login node to the cluster where users sign in and

develop, compile, and submit their applications for execution on the compute nodes. In a small cluster like QP, it is generally affordable to load additional server services on the same head node, such as compute node deployment services, DHCP, NAT gateway, batch system and scheduler, large volume file server (/home and /scratch), and a host-based InfiniBand subnet manager. Initially we planned for a single high-performance file server to serve the parallel file system over InfiniBand. However, few applications have been able to stress the server's performance and we now estimate that dividing the I/O tasks across two or three hosts will be a better solution.

Table 1 summarizes the performance characteristics of the different types of processors used in the cluster. Thus, 96% of the theoretical peak performance in single-precision floating-point arithmetic comes from 64 GPUs, 2.6% of the theoretical peak performance comes from 64 CPU cores, and 1.4% of the theoretical peak performance comes from 16 FPGAs. Double-precision floating-point arithmetic is not supported by the GPUs currently used in the cluster.

Processor type	*# of cores*	*Frequency*	*Attached memory*	*Attached memory bandwidth*	*On-chip memory*	*Peak GFLOPS*
AMD Opteron dual-core 2216	2	2.4 GHz	4 GB DDR2	10.7 GB/s	2x1 MB (L2 cache)	19.2 (DP)
NVIDIA G80GL	128	1.35 GHz	1.5 GB GDDR3	76.8 GB/sec	16x16 KB (multiprocessor shared memory)	346 (SP)
Xilinx Virtex-4 LX100	N/A	200 MHz	16 MB SRAM	6.4 GB/sec	0.5 MB (BRAM)	6.4 (DP) 20 (SP) 64 (int16)
			512 MB SDRAM	3.2 GB/sec		

Table 1. Characteristics of types of processors/memory used in the cluster.

2.1 Multi-core CPU host nodes

When selecting a host system for the compute nodes, it was challenging to find motherboards with two PCIe x16 slots available that will not degrade to 8x speeds when both are being used. The search narrowed further as we found that even on some approved chipsets, the BIOS on the board was not capable of supporting multiple GPUs. The HP xw9400 workstation [4] met all of our system requirements. It contains two 2.4 GHz AMD Opteron dual-core 2216 CPUs with 1 MB of L2 cache and 1 GHz HyperTransport link, has 8 GB (4x2 GB) of DDR2-667 memory and 80 GB SATA 7200 disk. But most importantly, it has two PCIe x16 slots that remain at 16x speed even when both are used, a PCIe x8 slot for an InfiniBand adapter, and a PCI-X slot for other uses, such as the FPGA accelerator card.

The compute nodes run Red Hat Enterprise Linux 5. GNU C/Fortran and Intel C/Fortran compilers are used to compile the CPU side of the applications.

2.2 GPU accelerators

The NVIDIA Quadro FX 5600 GPU cards [5] in the QP installation are physically located in NVIDIA Quadro Plex model IV deskside visual computing systems (VCSs) [6], two GPU cards per VCS. VCSs provide power and cooling to the cards whereas each compute node host has control and data connection to the four GPU cards through two x16 small form factor PCIe cards plugged into two of the host's PCIe

slots and two interconnect cables. In this configuration, two Quadro FX 5600 cards are attached to a single PCIe slot in the host node, allowing four GPUs per compute node.

The NVIDIA GPU driver is used to interface with the GPU cards. The CUDA 2.0 SDK [7] is currently used to develop applications for the GPU accelerators.

2.3 FPGA accelerators

The Nallatech H101-PCIX FPGA accelerator board [8] includes a Xilinx Virtex-4 LX100 FPGA chip, 16 MB SRAM, and 512 MB SDRAM. The card connects with the host system via a 133-MHz PCI-X interface.

Nallatech's field upgradeable systems environment (FUSE) API [9] is used to interface with the card. Application development for the accelerator card is carried out using the DIMEtalk application development environment [10]. DIMEtalk's network editor is used to describe the connectivity of various hardware modules used by the application, such as PCI-X interface and off-chip memory banks. The actual FPGA design is implemented as a DIMEtalk network module using DIME-C programming language. Nallatech's DIME-C C to VHDL Function Generator is used to translate DIME-C C code to VHDL, and DIMEtalk System Design is used to generate VHDL code and user constraints files for the network from the DIMEtalk network. The Xilinx Integrated Software Environment (ISE) [11] is used to synthesize and implement the design for the FPGA.

2.4 Network

Each cluster node is equipped with one integrated Broadcom 5755 NetXtreme Gigabit PCIe card and one Mellanox MHGS18-XTC PCIe x8 InfiniHost III Lx single-port 20 Gb/s 4X InfiniBand Host Channel Adapter (HCA) card [12]. A Netgear Prosafe 24-port Gigabit switch [13] is used for the Gigabit Ethernet network, and a 24-port Topspin 120 Server InfiniBand switch [14] is used for the InfiniBand-based network.

The OpenFabrics Alliance (OFA) OpenFabrics Enterprise Distribution (OFED) software stack [15] is used for communication.

2.5 Power and cooling

Measured power consumption of a single compute node with all four GPUs in use is about 1 kW. The Nallatech FPGA board's power consumption is 25 Watts [8]. Thus, we estimate that the total cluster's power consumption is under 18 kW. The cluster nodes and VCS units are air-cooled from front to back using internal fans. The cluster is located such that the front of the nodes draw chilled air from the "cold" aisle and exhaust air out the back to the "hot" aisle.

2.6 Cluster software

Cluster software on QP is similar to other Linux clusters. Users are not granted access to the computational nodes except for running their jobs, shared file systems exist for /home and /scratch, and some applications are not available on the compute nodes. Centralized authentication is used, user-based ssh keys, and compute nodes are on private subnet. The cluster nodes run Red Hat Enterprise Linux 5. The OFED software stack [15] is used for cluster nodes communication. Torque (aka PBS) batch system [16] and Moab scheduler [17] are used for job management. Lustre parallel file system [18] is used. System Imager [19] is used as the compute node image deployment tool. The only accelerator-specific software tools deployed on the cluster are NVIDIA GPU driver, CUDA 2.0 SDK [7], and Nallatech FUSE drivers and API [9].

However, we have had to recognize and deal with some specific challenges related to accelerators in a shared distributed environment. Complications arose when we configured the job system to share

node resources among users for greater utilization. The Torque batch system considers a CPU core as an *allocatable resource*, but it has no such awareness for any type of accelerators. The node property feature can be used to some extent to allow users to acquire nodes with the desired resources, but that does not prevent users from interfering with each other when sharing the same node. For CPU cores, the kernel scheduler makes the effort to balance the resources but GPU cores get no such benefit. With basic scheduling configurations, users cannot be prevented from launching more processes than they have allocated to them, filling up memory, or consuming too much of an I/O resource (network or disk). This is normally not a problem, as most users are aware of these issues and treat the resources carefully. However, in the case of GPUs, it turned out to be fairly easy for even 'well-behaved' users to interfere with each other because all users on a workstation can see and access all GPU devices. To mitigate this problem, we pre-load a library that we created to intercept CUDA device management functions and allow the user to see only the GPU devices that they have requested. Since Torque has no awareness of GPUs, users 'request' GPU resources in a different way. Since we have a 1:1 ratio of CPUs to GPUs and since Torque does support CPU resource requests, we decided to allocate as many GPUs as CPUs requested. An alternative approach may be to use the *consumable resources* attribute supported by Moab, where Moab tracks consumption of arbitrary resources.

We also discovered that due to the NUMA (Non-Uniform Memory Architecture) architecture of our host workstations, different GPUs had better memory bandwidth performance to the host CPU cores depending on what CPU socket the process is running on. The effective bandwidth difference can be as high as 6-8%. We created a file on each node containing the optimal mapping of GPU to CPU cores, and extended the CUDA device management wrapper library to set process affinity for the CPU cores closer to the GPU in terms of NUMA performance. The wrapper library also tries to simplify the user experience when running multiple processes by allowing each process to identically request GPU device0, yet have each process end up balanced across whatever physical GPUs are available. This is done by shifting the virtual GPU to physical GPU mapping by one each time a process is launched. The user's application sees only the virtual GPU devices, where device0 is rotated to a different physical GPU after any GPU device open call.

Prior to job execution, it is not uncommon to run pre-job node health checks on production Linux clusters. Log checks, current mounted file systems, and memory bandwidth are just a few examples of such tests. For GPUs, we also need to add checks to be sure the GPU clock is running at full speed, as there have been power-saving bugs in different drivers which cause the clock to get 'stuck' at a lower rate until the node is rebooted. We also realized that the Linux kernel cannot be depended upon to clean up or secure memory that resides on the GPU board. This poses security vulnerabilities from one user's run to the next, so we run a memory scrubbing utility between jobs on each node, which doubles as a memory health check for the GPU board.

3 Applications

While several research groups have been developing applications for the cluster [3], only a few of them managed to scale up their code to run on multiple nodes with accelerators. Table 2 lists a few of the applications implemented to date. NAMD (molecular dynamics) and WRF (weather modeling) are two examples of traditional HPC applications that have been recently re-designed to take advantage of GPU accelerators. TPACF (cosmology data analysis) is a new code implemented to take advantage of both FPGAs and GPUs.

The TPACF code is an interesting test case that has been implemented to take advantage of both FPGAs and GPUs. Its computational core is a subroutine that computes auto- or cross-correlation between datasets. This subroutine is called hundreds of times with datasets consisting of hundreds of

thousands to millions of data samples. Thus, the code naturally maps into the cluster's architecture: It is parallelized across multiple cluster nodes and its kernel is ported to the accelerators. The cross-node parallelization is implemented using MPI, FPGA kernel implementation is done using DIME-C, and GPU kernel is implemented in CUDA. A single binary is compiled and it can be executed using either FPGAs or GPUs. Different kernel implementation strategies are applied for the two accelerators:

- The FPGA kernel is designed closely following the original microprocessor implementation: Angular distances between data points are computed and mapped into bin counts. In contrast to the CPU implementation, which uses a *while* loop to search for the bin to which a given angular distance belongs, the FPGA implementation unrolls the *while* loop using *if-else* statements. This design allows for full pipelining of the computational kernel. Similar to the microprocessor implementation, the FPGA kernel uses double-precision floating-point arithmetic.
- The GPU implementation of the kernel consists of two parts: An angular distance kernel that computes all the angular distances between the pairs of data points and bin mapping kernel that maps the computed angular distances into the corresponding bins. The angular separations computed by the first kernel are stored in the GPU memory and used by the second kernel. This kernel partitioning is done to minimize the impact of branch divergence that leads to load imbalance of the GPU threads. In contrast to the microprocessor implementation, the GPU kernel is limited to single-precision floating-point arithmetic due to the GPU hardware limitations. This limits the use of the code for angular separations below 1 arcmin.

Application	# of CPU cores used	# of GPUs used	# of FPGAs used	non-accelerated application performance	accelerated application performance	overall speedup achieved
NAMD (molecular dynamics) [20]	60	60		0.471 sec/step	0.085 sec/step	5.5×
WRF (weather prediction) [21]	48	48		79.9 sec	64.9 sec	1.23×
TPACF (cosmology) [22]	8		8	5,537.7 sec	880.8 sec	6.3×
TPACF (cosmology) [22]	32	32		1,418.2 sec	29.3 sec	48.4×

Table 2. Applications developed to run on the multi-core/GPU/FPGA cluster.

Current TPACF implementation does not allow the simultaneous use of multiple types of accelerators. Future work includes implementing the application using Phoenix run-time (see next section) to enable the maximum utilization of the computational resources.

Unfortunately, Amdahl's law limits accelerator speedup in multiple ways: i) with the kernel running faster because of an accelerator, the rest of the application takes a higher percentage of the run-time; ii) partitioning the work among accelerators will eventually decrease the individual job size to a point where the overhead of calling an accelerator dominates the application execution time; and iii) load imbalance between nodes of the cluster due to imperfect work partitioning. As the result, a substantial code re-design and kernel tuning are required to take advantage of the application accelerators.

4 Phoenix application development framework

We are developing a run-time system called Phoenix to simplify the use of accelerators in distributed heterogeneous systems such as the QP cluster. We see three primary obstacles to delivering the

potential of such systems: i) effective utilization of resources across heterogeneous processing elements, ii) enabling legacy applications and programming models to leverage the emerging architectures, and iii) incompatibility of popular development tools with accelerators. We believe an intelligent, adaptive run-time system capable of efficient scheduling of tasks on heterogeneous processors coupled with a feature-rich set of debugging and performance tools is essential to efficient use of these resources. Porting legacy applications to these architectures can be accomplished via high-performance implementations of popular programming models and their associated tools layered atop this run-time platform.

Phoenix provides a modular and extensible architecture that parallel application developers, compiler writers, and system researchers can use as a vehicle for researching parallel scientific computing on heterogeneous platforms and as a fully optimized run-time framework on such systems. The layering of existing popular parallel programming paradigms, such as Message Passing Interface (MPI), atop this system enables many existing applications to execute on these systems with little or no modification. Programming extensions through the use of compiler directives and source-to-source transformation tools allow legacy applications to further leverage new features.

Phoenix provides an essential foundation upon which best strategies and practices for optimal utilization of heterogeneous resource can be researched, tested, and deployed. Our primary objective is to research and provide an implementation of such a run-time system. The system is modular so that strategies related to scheduling, memory management, etc., can be plugged in and tested for suitability to particular hardware resources and to particular applications. As part of Phoenix development, we are implementing a wide range of system-level resource management strategies, as a starting point for discovering the best practices for parallel application execution on heterogeneous platforms.

The overall application development environment, called the Phoenix ecosystem, consists of the following distinct components:

- Run-time System: This is the primary software layer. It provides a rich feature set for efficient layering of parallel programming model at one end and functionality for optimal utilization of hardware resources at the other end. The core functionality of the Phoenix ecosystem for scheduling, thread management, memory allocation, debugging, etc., is contained in this layer.
- Tools: Standard performance tools (KOJAK, TAU and PMPI) and debugging tools (Purify and Electric Fence) will be integrated with the Phoenix run-time system and retrofitted to be used across various accelerator technologies. This will involve interfacing debugging and performance monitoring capabilities of accelerators such as the NVIDIA CUDA GPUs to these standard tools.
- Parallel Programming Paradigm: Layering of a parallel programming language or library over Phoenix to test and validate run-time systems and tools with current production quality parallel applications.

We currently have a basic prototype for the core Phoenix run-time on QP. We have also implemented MPI 1.2 on top of this run-time for a proof of concept and preliminary testing and validation of Phoenix run-time with MPI parallel applications [23].

Acknowledgements

The cluster was built with support from the NSF CRI (CNS 05-51665) and NSF HEC Core Facilities (SCI 05-25308) programs along with generous donations of hardware from NVIDIA, Xilinx, and Nallatech. Accelerator implementations of TPACF application were supported by the NSF STCI (OCI 08-10563) and NASA AISR (NNG06GH15G) programs. Special thanks to Trish Barker from NCSA's Office of Public Affairs for help in preparing this publication.

Bibliography

[1] GPU Cluster overview, http://www.ncsa.uiuc.edu/Projects/GPUcluster/

[2] W. Hwu, D. Kirk, ECE 498 AL1 : Programming Massively Parallel Processors, http://courses.ece.uiuc.edu/ece498/al1/

[3] GPU Cluster Projects, http://www.ncsa.uiuc.edu/Projects/GPUcluster/projects.html

[4] HP xw9400 Workstation QuickSpecs, http://h18000.www1.hp.com/products/quickspecs/12594_na/12594_na.PDF

[5] NVIDIA Quadro FX 5600 datasheet, http://www.nvidia.com/docs/IO/40049/quadro_fx_5600_datasheet.pdf

[6] NVIDIA Quadro Plex visual computing system, http://www.nvidia.com/content/quadroplex/nvidia_quadroplex_product_overview_Jan08_.pdf

[7] NVIDIA Compute Unified Device Architecture (CUDA) ver. 2.0 Programming Guide, http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf

[8] H100 Series FPGA Application Accelerators, http://www.nallatech.com/mediaLibrary/images/english/5595.pdf

[9] Field Upgradeable Systems Environment (FUSE), http://www.nallatech.com/mediaLibrary/images/english/2343.pdf

[10] DIMEtalk V3.0 Application Development Environment, http://www.nallatech.com/mediaLibrary/images/english/4613.pdf

[11] ISE Design Suite 10.1 – ISE Foundation, http://www.xilinx.com/publications/prod_mktg/pn0010867.pdf

[12] Mellanox HCA Cards, http://www.mellanox.com/pdf/products/hca/IH3LX.pdf

[13] Netgear Smart Switches, http://www.netgear.com/Products/Switches/SmartSwitches.aspx

[14] Topspin 120 Server Switch datasheet, http://www.topspin.qassociates.co.uk/literature-library/datasheets/120-datasheet.pdf

[15] OpenFabrics Enterprise Distribution (OFED), http://www.openfabrics.org/download_linux.htm

[16] TORQUE Resource Manager, http://www.clusterresources.com/pages/products/torque-resource-manager.php

[17] Moab Workload Manager, http://www.clusterresources.com/pages/products/moab-cluster-suite/workload-manager.php

[18] Lustre file system, http://wiki.lustre.org/

[19] System Imager, http://wiki.systemimager.org/

[20] J. Phillips, J. Stone, and K. Schulten, Adapting a message-driven parallel application to GPU-accelerated clusters. In Procs International Conference for High Performance Computing, Networking, Storage and Analysis (SC08), 2008.

[21] J. Michalakes, and M. Vachharajani, GPU Acceleration of Numerical Weather Prediction, Parallel Processing Letters, vol. 18, no. 4, pp 531-548, 2008.

[22] V. Kindratenko, D. Roeh, A. Martinez, G. Shi, R. Brunner, Investigating Application Analysis and Design Methodologies for Computational Accelerators, NCSA Technical Report, 2009.

[23] A. Pant, H. Jafri, V. Kindratenko, Phoenix: A Runtime Environment for High Performance Computing on Chip Multiprocessors, in Procs 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2009).